REMARKS

Prior to this Response, claims 1-6, 8-13, 20 and 22 were pending in the Application. No claims have been added, canceled, or amended in this Response. Thus, after entry of this Response, claims 1-6, 8-13, 20, and 22 remain for consideration by the Examiner.

Examiner Interview

Initially, Applicant thanks the Examiner for the time spent discussing the application in an Examiner Interview conducted on September 1, 2010. During the interview, the Examiner and Applicant's attorney, Libby Huskey, discussed the rejections under 35 U.S.C. § 112, first paragraph, as well as the rejections under 35 U.S.C. § 103. Specifically, the interview included a discussion of the portions of the original specification that Applicant believes provide support for the claims as presented. In addition, the interview included a discussion regarding Applicant's position that the combination of U.S. Patent No. 7,171,663 ("Moore"), U.S. Patent No. 6,370,590 ("Nitz"), and U.S. Patent No. 7,325,233 ("Kuck") fails to teach one or more elements of each of the independent claims 1, 12, and 20 and further that the references have been improperly combined. The Examiner indicated that Applicant's arguments have merit and that that she would like to take them under advisement. Thus, the Examiner requested that Applicant submit an arguments-only response to the final Office Action detailing Applicant's arguments as discussed in the interview. Applicant has set forth these arguments below.

Claim Rejections Under 35 U.S.C. § 112

Claims 1-6, 8-13, 20 and 22 stand rejected under 35 U.S.C. § 112, first paragraph, as failing to comply with the written description requirement. The Examiner specifically argues that though the specification teaches concurrent tasks and how they operate, the specification does not indicate how concurrent determination and manipulation occur. Applicant disagrees.

Figure 1B of the original specification shows that a virtual machine 112 receives several platform-specific events E1, E2, E3, and E4 *in parallel*. Paragraphs [0027]-[0029] reference Figure 1B in explaining that once the virtual machine receives the platform-specific events E1, E2, E3, and E4, a smart event-dispatcher 118 within the virtual machine 112 determines which of two tasks 120, 122 operating on the virtual machine 112 should receive and process each of the platform-specific events E1, E2, E3, and E4. This determination may be based on a number of criteria associated with the platform-specific events (e.g., events may be pre-assigned to a particular task, delivered to a task that is currently in the foreground, dispatched to tasks according to user selections, dispatched according to event dispatching logic running on the smart event-dispatcher, etc.).

Paragraphs [0041]-[0042] go on to explain that because tasks (e.g., the tasks 120, 122 of Figure 1B) that are implemented in Java may not be able to access event data structures that are implemented in another programming language, the virtual machine (e.g., the virtual machine 112 of Figure 1B) may be implemented in languages other than Java to manipulate data structures for the platform-specific events (e.g., events E1, E2, E3, and E4) to represent the events as Java data structures. Once manipulated, the tasks operating on the virtual machine (e.g., the tasks 120, 122 of Figure 1B) may access the data structures of the platform-specific evens for processing.

It is Applicant's position that Figure 1B clearly shows the external events E1, E2, E3, and E4 entering the virtual machine 112 and being received at the smart event-dispatcher 118 at the same time, or simultaneously (not serially and/or sequentially). It follows that the smart event-dispatcher also makes a parallel, or concurrent, determination regarding which task will process each of the external events, and further, that the virtual machine simultaneously manipulates each of the events to comply with a data structure format supported by the selected task. Thus, Applicant submits that the combination of Figure 1B and the corresponding portions of the specification discussed above provide support for the claims as presented, or for simultaneously receiving a plurality of events and then concurrently/simultaneously

handling each event (i.e., determining which task should process each event and manipulating a data structure of the events to comply with the selected task).

With the above being said, Applicant understands the Examiner's argument that because the Written Description does not set forth specific temporal limitations regarding the receipt and handling of the external events E1, E2, E3, and E4 shown in Figure 1B, the specification could also be understood to provide support for receiving and handling the external events at the virtual machine serially, rather than simultaneously. Thus, because the specification clearly does not limit the receipt and handling of external events to receiving and handling events serially, the specification can be read to provide support for *both* of the following: (1) simultaneously receiving a plurality of platform-specific events, concurrently determining a selected task that should process the platform-specific events, and simultaneously manipulating the events received by modifying a data structure of each event to comply with a structure supported by the selected task, and (2) serially receiving a plurality of platform-specific events, determining a selected task that should process the platform-specific events, and manipulating the events received by modifying a data structure of each event to comply with a structure supported by the selected task. If the specification supports both the simultaneous and serial receipt and handling of external events, Applicant believes that the narrowing amendment to require the former, or to require that the external events be received and handled simultaneously/concurrently, is supported by the specification. For instance, in a simple mechanical example, a drawing that shows three pipes flowing liquid into a tank supports concurrent or simultaneous flow of liquid into the tank unless the specification is specifically limited otherwise (e.g., only alternating flow managed by valves, etc.).

In addition, as stated in the Examiner Interview, Applicant would be willing to amend the specification to add the narrowing term "simultaneously" where appropriate if the Examiner believes such an amendment would be helpful. Moreover, the Examiner has indicated an opinion that support for "concurrently" determining and "simultaneously" manipulating may be more tenuous than the support for "simultaneously" receiving. While Applicant believes that support for each of these modifiers is provided, as discussed above, Applicant would be willing to delete the

modifiers "concurrently" and "simultaneously" before the terms "determining" and "manipulating," respectively.

Considering the above, Applicant believes that the specification fully supports the invention as claimed and asks that the rejection under 35 U.S.C. § 112 be withdrawn. If the Examiner would like to discuss possible amendments in relation to the § 112 rejection, Applicant invites the Examiner to contact the undersigned.

Claim Rejections Under 35 U.S.C. § 103

Claims 1-6

In the final Office Action mailed June 21, 2010, claims 1-6 were rejected under 35 U.S.C. § 103(a) as being unpatentable over the combination of Moore, Nitz, and Kuck. Applicant initially requests reconsideration because the proposed combination, even if proper, does not teach each element presented by the combination of features required by claim 1. Specifically, and as discussed above, claim 1 requires the step of "simultaneously receiving, by the virtual machine, a plurality of platform-specific events which are associated with the first platform." As discussed in paragraph [0008]-[0010] of the original specification, conventional virtual machines do not provide a multi-tasking environment in a manner that allows a user to concurrently interact with tasks that receive input from the user or another source. That is, while conventional computing systems support multi-tasking and allow multiple tasks to be performed concurrently (e.g., a user can use a calculator application and a music player concurrently to perform calculations while listening to music), conventional virtual machines do not provide a multi-tasking environment that allows a user to interact with two concurrent tasks. In other words, concurrent tasks running on conventional virtual machines cannot receive and process simultaneously received external events (e.g., inputs from a user keyboard, mouse, etc.).

While the Examiner initially admits that Moore does not teach a virtual machine that receives external events simultaneously, the Examiner appears to backtrack somewhat and later argues that Figure 1 of Moore "shows multiple external events coming in VM 140" and that Moore teaches that "requests, or events, may be

processed simultaneously (Column 6, lines 50-53)." Final Office Action, page 4, lines 12-13; page 10, section (i).

First, with respect to Figure 1 of Moore, Applicant notes that Figure 1 shows three external requests/events 115, 120, 125 in transit from clients 105, 110 to a network 130. The requests 115, 120, 125 are not shown entering a virtual machine 140. Indeed, it is unclear from Figure 1 how the requests 115, 120, 125 are distributed to a server 135 and then to the virtual machine 140, and there is no indication that the requests are received simultaneously by the virtual machine 140. With this in mind, an examination of the Moore specification lends further meaning to Figure 1 and makes it clear that the virtual machine 140 does not receive the external requests 115, 120, 125 simultaneously. Moore is directed to a utility for interrupting external event processing in server-side programs. Specifically, the Background section of Moore explains that one problem with conventional virtual machines is that virtual machine servlets process client requests in a manner that requires the client to wait until a servlet has processed or completed a request with no way to interrupt or otherwise modify a process executing within the servlet before completion. Moore, column 1, lines 39-42. This is inconvenient in the event that the request becomes moot or a subsequent request merits elevated priority. See Moore, column 1, lines 43-55.

To resolve this problem, Moore teaches a method for handling external interrupts within a servlet or server-side program instance. An exemplary method is shown in Figure 2 and discussed in column 6, line 12 – column 7, line 41 of Moore. These portions of Moore show and explain that an external request is received (step 202) and relayed to the virtual machine (step 204) before a thread is initialized for processing the request (step 210). Once the thread has begun execution (step 214), the method can search for additional external requests that require processing (step 216). If more requests exist, the server can *then receive* another request from the client (step 218) and initialize a new thread for processing the subsequent request. Each thread may be associated with an event monitor (step 212), which can detect and react to certain events that occur in the virtual machine. For example, the event monitor can be configured to take a specified action upon the detection of a monitored event, such as interrupting or terminating a currently operating thread, saving server resources and

reducing client response time. Thus, the utility disclosed in Moore receives one request at a time. Notably, and as the Examiner correctly points out, the Moore system is capable of *simultaneously executing* multiple threads for external requests, but these threads are executing for external requests *received at varying times*. Moore, column 6, lines 43-53. Moore does not disclose *simultaneously receiving* external requests by the virtual machine but rather *simultaneously processing requests that are consecutively or serially received*.

The Examiner also cites Kuck as teaching a VM that may receive multiple external events simultaneously. Kuck differs from the claimed invention, however, in that claim 1 requires simultaneously receiving, by a virtual machine that concurrently supports first and second tasks, a plurality of platform-specific events. In contrast, Kuck discloses receiving multiple requests at a single program (i.e., task). Kuck, column 7, lines 18-29. This distinction is meaningful because rather than a virtual machine that simultaneously receives platform-specific events and assigns each of those events to one of at least two tasks supported by the virtual machine, Kuck involves repeatable process attachable virtual machines ("PAVMs"), or numerous virtual machines that each bind to a specific work process allocated to an associated server. Kuck, column 5, lines 13-46; Figure 2. Thus, Kuck does not disclose a virtual machine that supports first and second tasks and receives simultaneous requests for processing by a selected one of those tasks, but rather teaches a single program that is operated by one of many separate PAVMs, where the program is capable of receiving concurrent requests. Kuck, column 7, lines 18-29. As discussed in the Examiner Interview, if the Examiner has remaining questions or concerns regarding whether this distinction has been effectively clarified in the claims, Applicant welcomes the Examiner's input.

Moving to additional elements of claim 1, the Examiner admits that Moore fails to teach a centralized entity or role for determining which tasks should process each of the platform-specific events and manipulating the platform-specific events by modifying their data structures to be compliant with a data structure format supported by said selected task. Instead, the Examiner relies on Nitz for the limited purpose of showing "selecting one of the first and second tasks as a selected task for receiving the platform-specific event" and "manipulating the platform specific event by modifying its

data structure to be compliant with a data structure format supported by the selected task." Final Office Action, page 4.

During the Examiner Interview, both Applicant and the Examiner agreed that Nitz does not involve processing platform-specific events received at a virtual machine. Instead, Nitz involves facilitating communications, or message-passing, between sub-applications nested within a vertical application such that the sub-applications may communicate with each other when processing performed by one sub-application affects processing performed by another sub-application. Nitz, Background and Summary Sections.

Specifically, the cited portions of Nitz disclose attaching an "agent" to each sub-application. The agent is configured to translate messages between the message format supported by the sub-application and a common view (format X). Each agent is linked to other agents through a centralized broker that manages routing of messages between agents. Thus, when a sub-application begins executing, the agent attached to the sub-application registers with the broker by sending registration information to the broker. The registration information includes subscription criteria that the broker uses to determine which messages are to be routed to the agent. As sub-applications begin executing, the sub-applications transmit messages to their corresponding agents in the format supported by the sub-applications. The agents then translate the messages from that format to the format supported by the common view (format X) before those messages are transmitted or published to the broker. The broker evaluates the subscription criteria provided by the agents to determine which agents are to receive the messages. Each agent receiving the message translates the message from format X to a message formatted for the corresponding sub-application and passes the message to the corresponding sub-application to complete the process.

Applicant notes that in relation to the claimed invention, Nitz merely discloses a centralized module (i.e., the broker) that is used in passing and manipulating data that is transferred between two points (i.e., two agents associated with sub-applications), and Applicant strongly believes the Examiner has genericized the reference to an unacceptable degree. *Nitz does not disclose or even imply any determination regarding an appropriate task for processing an external event; nor does Nitz involve manipulating*

a data structure of an external event so that the structure is compatible with the selected task. Examiner's abstraction of Nitz implies that any reference disclosing a centralized translation function or module is appropriately combined with Moore and Kuck to disclose the claimed invention. If this were considered proper, any one of literally hundreds or thousands of computer networking references involving data translation could be combined with Moore and Kuck and used to reject claim 1, and Applicant submits that this is improper and that Nitz is simply too far removed from the claimed invention to be relevant.

For these reasons, Applicant believes that claim 1 is allowable over the cited combination of Moore, Nitz, and Kuck and asks that the rejection be withdrawn and the claim allowed. Because claims 2-6 depend from claim 1 (directly or indirectly), Applicant believes that these claims are patentable over the cited combination for at least the reasons provided with respect to claim 1.

Claims 12 and 13

Claims 12 and 13 also stand rejected under 35 U.S.C. § 103(a) as being unpatentable over the combination of Moore, Nitz, and Kuck. While independent claim 12 is written in apparatus form and is directed to a computer-implemented virtual machine, claim 12 includes limitations similar to those discussed above with respect to claim 1, and Applicant believes that claim 12 is patentable for at least the reasons provided for claim 1.

Further, independent grounds support the patentability of claim 12, which in relevant part requires (1) a platform-specific event dispatcher that operates to: (a) simultaneously receive a plurality of platform-specific events associated with a first platform; and (b) concurrently select, for each platform-specific event, a selected task that defines which of first and second tasks operating on the virtual machine should receive the platform-specific events; *(2) a platform-specific event-repository for the selected task, wherein the platform-specific event repository provides event storage prior to processing by the selected task; and (3) a platform-specific event-handler for the selected task, wherein the platform-specific event-dispatcher operates to place the*

platform-specific event in said platform-specific event-repository and invoke the platform-specific event-handler to initiate processing of the platform-specific event.

As explained in Paragraphs [0032]-[0035] and shown in Figure 2 of the original specification, one embodiment of a virtual machine 204 may include a smart event-dispatcher (i.e., a manager) 212. The smart event-dispatcher 212 may dispatch a specific-platform event E1 to a task 214 and a specific-platform event E2 to a concurrently supported task 216. In greater detail, the smart event-dispatcher 212 may place the platform-specific event E2 in an event-repository 220 and notify an event-handler 221 associated with the task 216 that an external, platform-specific event has been delivered. This invokes the event-handler 221 associated with the task 216 and initiates processing of event E2 on behalf of the task 216.

The Examiner relies on the language of claim 1 of Kuck for the limited purpose of teaching the platform-specific event-repository and the platform-specific event-handler. With respect to the event repository, claim 1 of Kuck requires receiving a request corresponding to a user session and "maintaining a request queue having the request, and bringing the request to a front of the request queue." Kuck, column 12, line 67 – column 13, line 2. In this regard, Kuck merely discloses maintaining and managing a centralized request queue for incoming requests, *not a platform-specific event-repository for the selected task* (i.e., the task operating on the virtual machine that has been selected to process the request). If anything, Kuck implies some type of common storage that is used to manage all incoming requests. Thus, Kuck fails to teach a "platform-specific event-repository for the selected task, wherein the platform-specific event repository provides event storage prior to processing by the selected task."

With respect to the platform-specific event-handler, claim 1 of Kuck discloses selecting an operating system process and binding a virtual machine to the selected operating system process to process a request. The "binding" of the virtual machine to the selected process includes mapping a portion of a shared memory associated with the virtual machine to a corresponding address space of the selected operating system process. Kuck, column 13, lines 6-13. This general language relating to processing a request within a virtual machine fails to disclose the specific elements required of claim 12. Indeed, claim 1 of Kuck discloses associating a shared memory with the operating

system selected to process the request. In contrast, claim 12 mandates task-specific storage and task-specific processing initialization, and Kuck simply fails to disclose the requisite platform-specific event-handler for the selected task, where the platform-specific event-dispatcher operates to place the platform-specific event in said platform-specific event-repository and invoke the platform-specific event-handler to initiate processing of the platform-specific event.

For these additional reasons, Applicant believes that independent claim 12 is patentable over the cited combination. Further, claim 13 depends from claim 12 and is patentable for at least the same reasons as claim 12.

Claims 20 and 22

Claims 20 and 22 also stand rejected under 35 U.S.C. § 103(a) as being unpatentable over the combination of Moore, Nitz, and Kuck. While independent claim 20 is written in apparatus form and is directed to a computer readable storage medium including a computer program for processing platform-specific events, claim 20 includes limitations similar to those discussed above with respect to claim 1, and Applicant believes that claim 20 is patentable for at least the reasons provided for claim 1. Further, claim 22 depends from claim 20 and is patentable for at least the same reasons as claim 20.

Conclusions

Based upon the foregoing, Applicant believes that all pending claims are in condition for allowance and such disposition is respectfully requested. In the event that a telephone conversation would further prosecution and/or expedite allowance, the Examiner is invited to contact the undersigned.

Please credit any overpayment or charge any underpayment to Deposit Account No. 50-1419.

Respectfully submitted,

MARSH FISCHMANN & BREYFOGLE LLP

Date: 9/8/10

By: Libby Huskey

Libby A. Huskey
Reg. No. 59,087
720-562-5507 Tel